(12) UK Patent Application (19) GB (11) 2 320 344 (13) A

(43) Date of A Publication 17.06.1998

(21) Application No 9720400.2

(22) Date of Filing 26.09.1997

(30) Priority Data
(31) 08722434 (32) 07.10.1996 (33) US

(71) Applicant(s)
International Business Machines Corporation

(Incorporated in USA - New York)

Armonk, New York 10504, United States of America

(72) Inventor(s)
Murthy Devarakonda
Ajay Mohindra
Deborra Jean Zukowski

(74) Agent and/or Address for Service
G M Zarbi
IBM United Kingdom Limited, Intellectual Property Department, Mail Point 110, Hursley Park, WINCHESTER, Hampshire, SO21 2JN, United Kingdom

(51) INT CL[6]
G06F 17/30

(52) UK CL (Edition P)
G4A AFGN AUDB

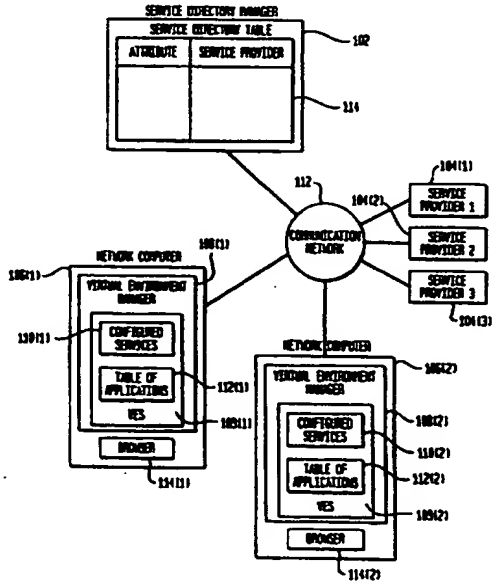
(56) Documents Cited
EP 0780778 A2 US 5483652 A

(58) Field of Search
UK CL (Edition P) G4A AFGN AUDB
INT CL[6] G06F 13/14 17/30
ONLINE: WPI, INSPEC, COMPUTER

(54) Providing on-demand access to network services for Network Computers

(57) Downloadable Virtual Environment Manager (VEM) middleware 108 enables applications executing on a network client such as a Network Computer 106 to access network services, including system services such as printing and local storage. The VEM includes a Virtual Environment Supervisor (VES) 109 which configures the default client services to set up the user's desktop, and has access to all services available on the network through a Service Directory Manager (SDM) 102 that maintains a Service Directory Table 114. An application executing on the Network Computer and wishing to use one of the services communicates with its local VEM. The VES checks the SDM to get the handle to the service (fig. 5) and, if the service has a stub, this is downloaded. The VEM then adds a service entry to Configured Services Table 110 that includes the service's handle and, if appropriate, a reference to the stub, so that the application can then use the service when required.

FIG. 1



GB 2 320 344 A

FIG. 1

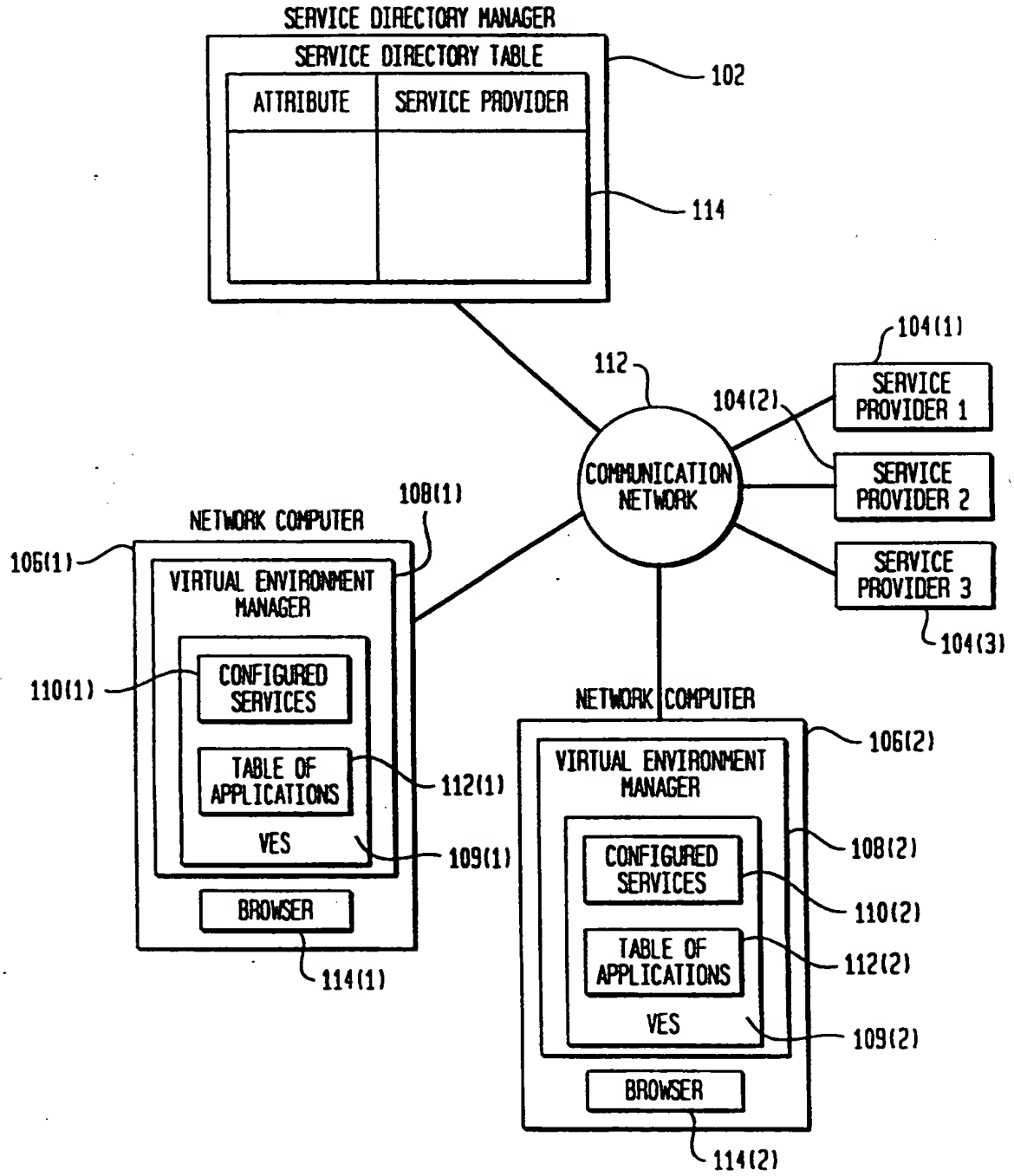

SERVICE DIRECTORY MANAGER
SERVICE DIRECTORY TABLE
ATTRIBUTE
SERVICE PROVIDER
102
114
112
104(1)
104(2)
SERVICE PROVIDER 1
SERVICE PROVIDER 2
SERVICE PROVIDER 3
104(3)
COMMUNICATION NETWORK
108(1)
NETWORK COMPUTER
106(1)
VIRTUAL ENVIRONMENT MANAGER
110(1)
CONFIGURED SERVICES
TABLE OF APPLICATIONS
112(1)
VES
109(1)
BROWSER
114(1)
NETWORK COMPUTER
106(2)
VIRTUAL ENVIRONMENT MANAGER
108(2)
CONFIGURED SERVICES
110(2)
TABLE OF APPLICATIONS
112(2)
VES
109(2)
BROWSER
114(2)

FIG. 2

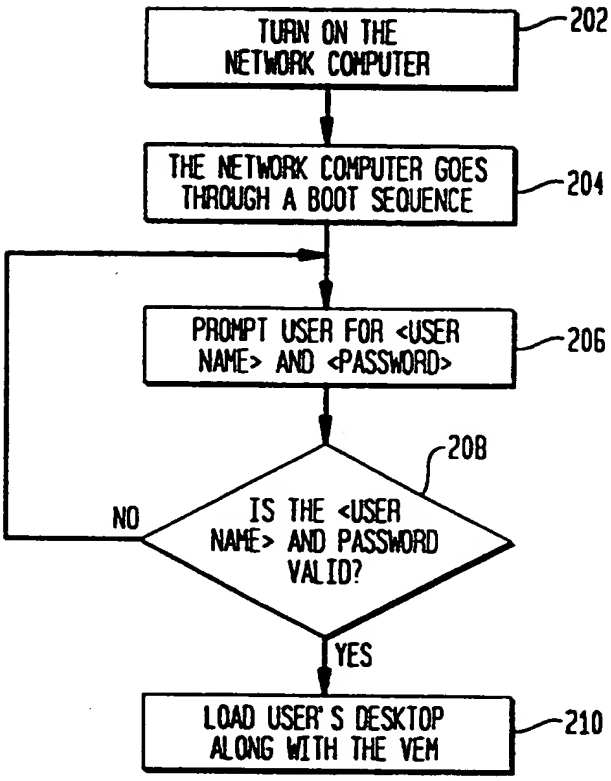

TURN ON THE NETWORK COMPUTER
202
THE NETWORK COMPUTER GOES THROUGH A BOOT SEQUENCE
204
PROMPT USER FOR <USER NAME> AND <PASSWORD>
206
208
IS THE <USER NAME> AND PASSWORD VALID?
NO
YES
LOAD USER'S DESKTOP ALONG WITH THE VEM
210

FIG. 3

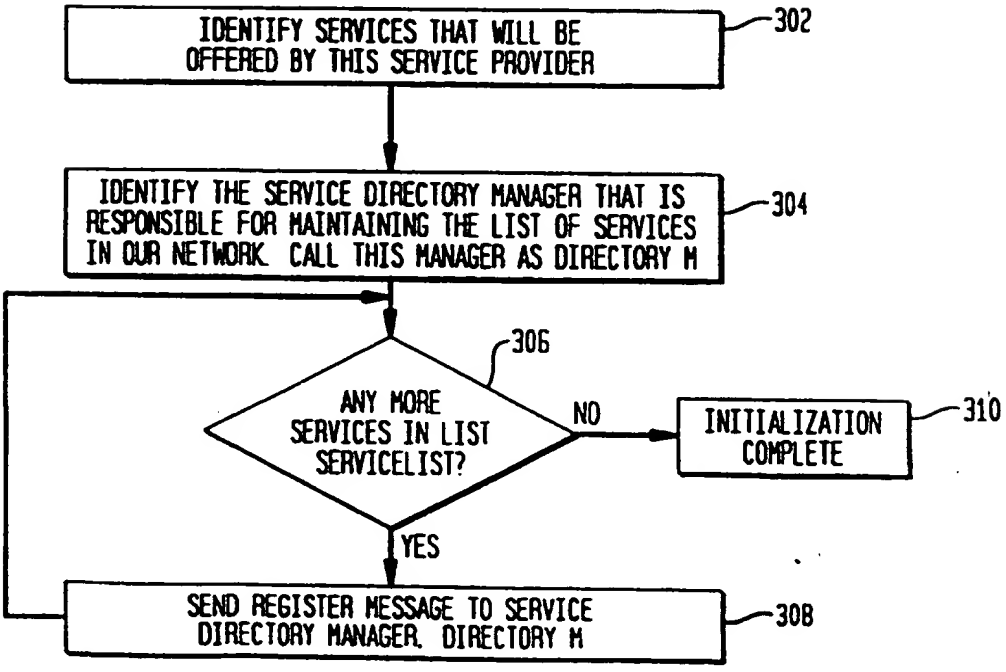

IDENTIFY SERVICES THAT WILL BE OFFERED BY THIS SERVICE PROVIDER
302
IDENTIFY THE SERVICE DIRECTORY MANAGER THAT IS RESPONSIBLE FOR MAINTAINING THE LIST OF SERVICES IN OUR NETWORK. CALL THIS MANAGER AS DIRECTORY M
304
306
ANY MORE SERVICES IN LIST SERVICELIST?
NO
INITIALIZATION COMPLETE
310
YES
SEND REGISTER MESSAGE TO SERVICE DIRECTORY MANAGER. DIRECTORY M
308

FIG. 4

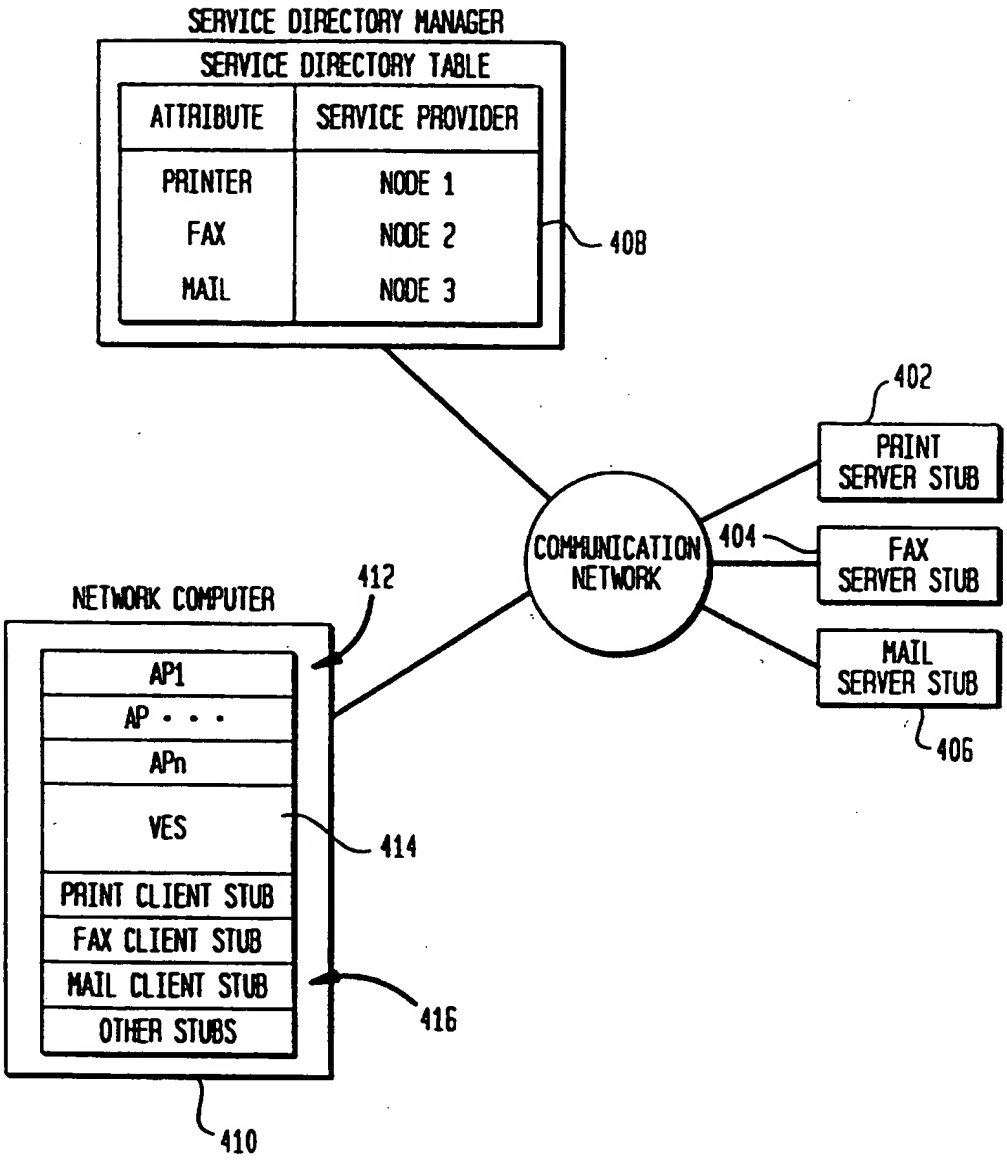

SERVICE DIRECTORY MANAGER
SERVICE DIRECTORY TABLE
ATTRIBUTE
SERVICE PROVIDER
PRINTER
NODE 1
FAX
NODE 2
MAIL
NODE 3
408
402
PRINT SERVER STUB
COMMUNICATION NETWORK
404
FAX SERVER STUB
MAIL SERVER STUB
406
412
NETWORK COMPUTER
AP1
AP • • •
APn
VES
414
PRINT CLIENT STUB
FAX CLIENT STUB
MAIL CLIENT STUB
OTHER STUBS
416
410

FIG. 5

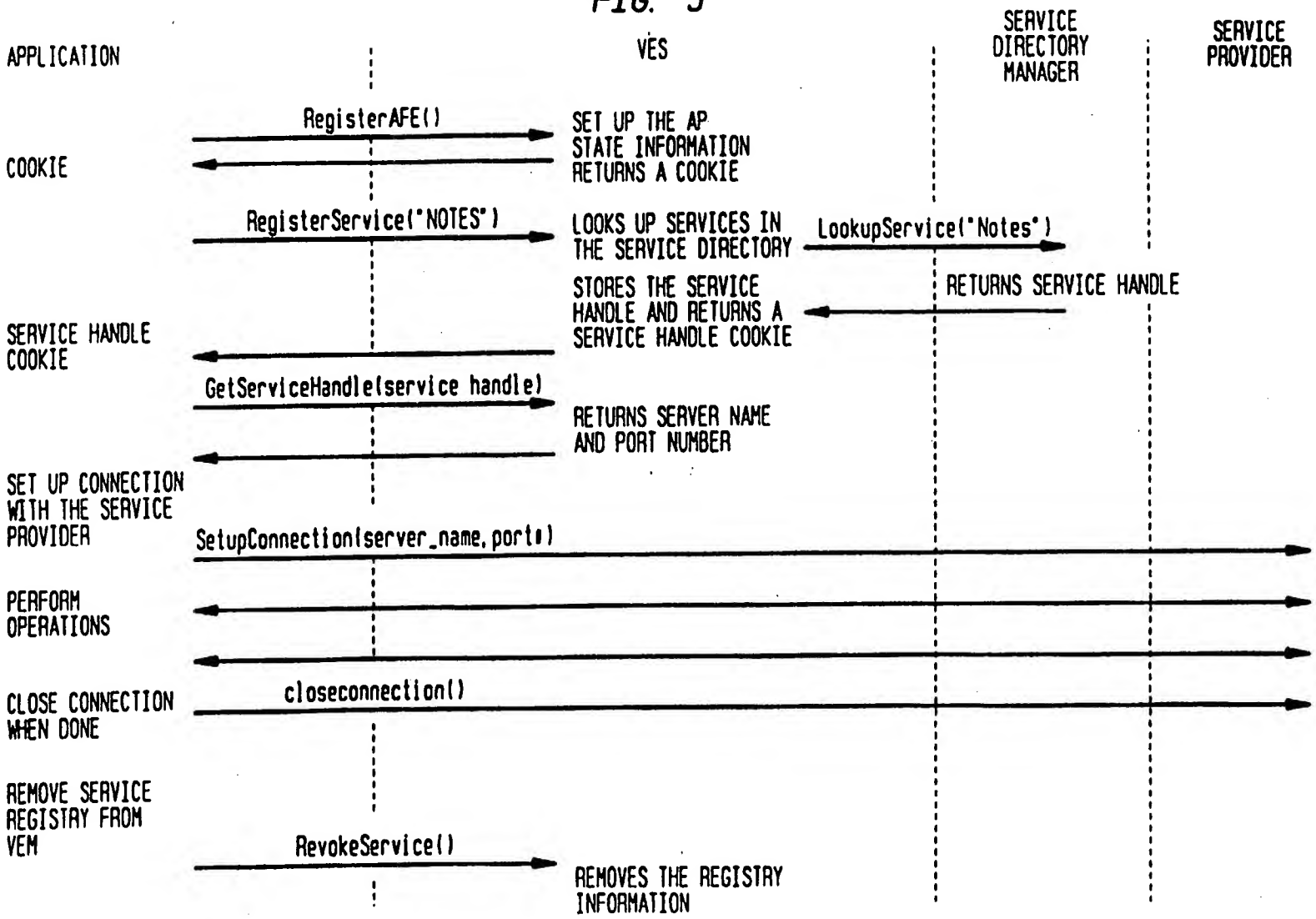
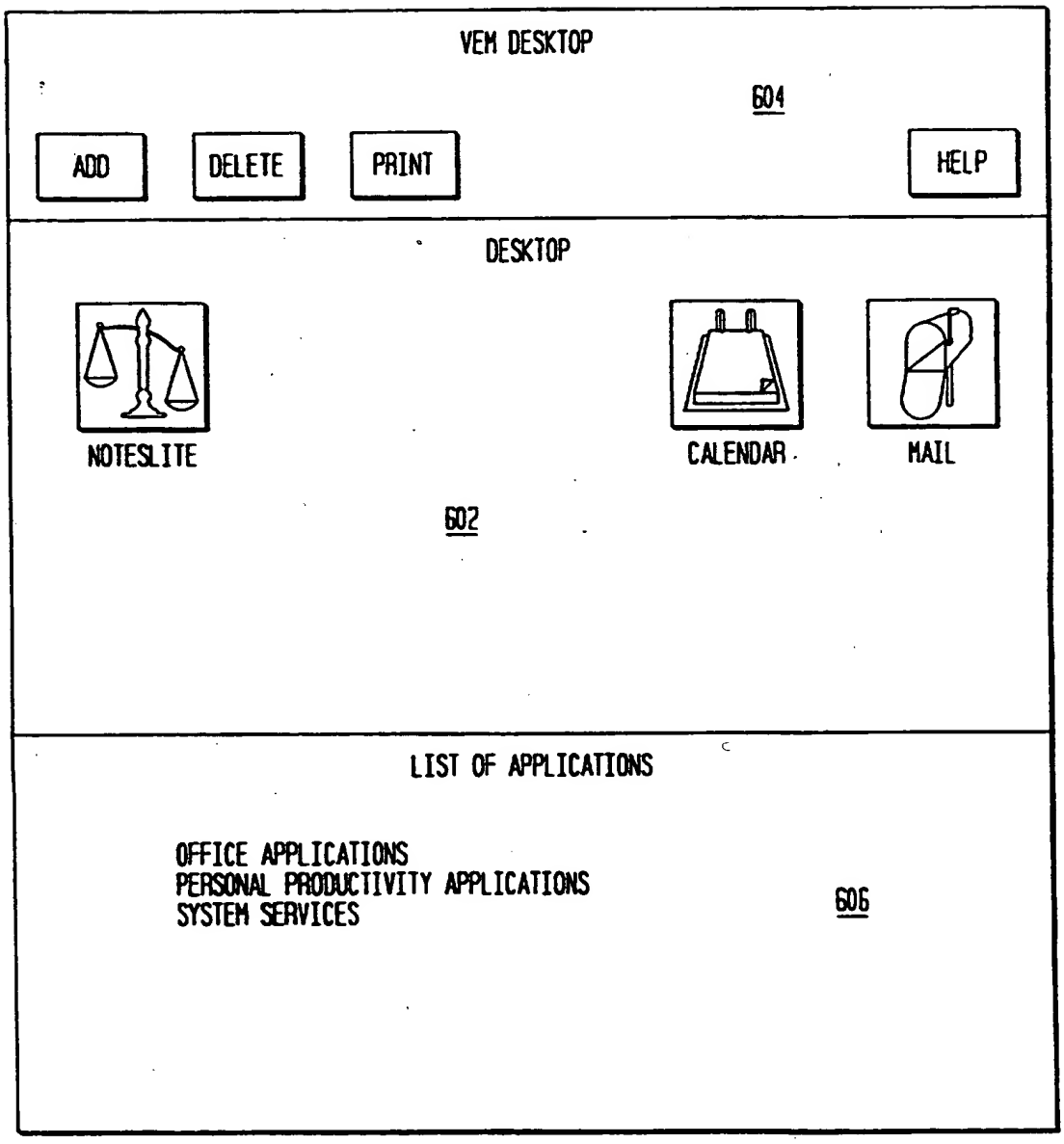

APPLICATION
VES
SERVICE DIRECTORY MANAGER
SERVICE PROVIDER
RegisterAFE()
SET UP THE AP STATE INFORMATION RETURNS A COOKIE
COOKIE
RegisterService("NOTES")
LOOKS UP SERVICES IN THE SERVICE DIRECTORY
LookupService("Notes")
STORES THE SERVICE HANDLE AND RETURNS A SERVICE HANDLE COOKIE
RETURNS SERVICE HANDLE
SERVICE HANDLE COOKIE
GetServiceHandle(service handle)
RETURNS SERVER NAME AND PORT NUMBER
SET UP CONNECTION WITH THE SERVICE PROVIDER
SetupConnection(server_name, port#)
PERFORM OPERATIONS
CLOSE CONNECTION WHEN DONE
closeconnection()
REMOVE SERVICE REGISTRY FROM VEM
RevokeService()
REMOVES THE REGISTRY INFORMATION

FIG. 6


VEM DESKTOP
604
ADD
DELETE
PRINT
HELP
DESKTOP
NOTESLITE
CALENDAR
MAIL
602
LIST OF APPLICATIONS
OFFICE APPLICATIONS
PERSONAL PRODUCTIVITY APPLICATIONS
SYSTEM SERVICES
606

2320344

VIRTUAL ENVIRONMENT MANAGER FOR NETWORK COMPUTERS

FIELD OF THE INVENTION

This invention relates to network computing. More specifically, this invention relates to methods and means for providing access to network services (for example, system services such as printing and local storage) to applications executing on network connected computers.

BACKGROUND ART

As the network computing paradigm becomes ubiquitous, simplified inexpensive desktop computers, known as the Network Computers, that have no means of independent existence will become common place. Such devices may run a low function operating system (for simplicity) and rely on servers for basic system services including local storage, printing, and monitoring. While traditional clients, such as PCs, provide their own basic system services they are also likely to seek additional services from the network.

Existing approaches used to provide these basic system services to applications include the existence of a full-feature operating environment on the Network Computer and/or the requirement that each application provide its own set of needed services. The former approach is not feasible for network clients because these computers are not necessarily equipped with adequate physical resources such as physical memory and attached peripheral devices (e.g., disk drives) to support a full-feature operating environment (e.g., in the case of a Network Computer). The latter approach has the drawback of making each application aware of the platform and environment it can be run under so that it can provide support for necessary basic system services. Further, the former approach adds to the cost of the network clients, while the latter approach adds to the complexity in the design of applications.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a technique which alleviates the above drawbacks.

According to the present invention we provide a method for providing access to network services to an application executing on a network

client, comprising the steps of: determining when the application requires access to a remote network service; in response to a determination that the application requires access to the remote network service: locating the remote network service required by the application on demand; and downloading at least one object into a memory of the network node that enables the application to access the remote network service on demand.

Further according to the present invention we provide a method for providing on-demand access to system services to applications executing on a network computer, comprising the steps of: a) in response to a demand from a user, downloading a virtual environment manager that provides access to remote network services; b) identifying a set of system services; c) configuring the set of system services by contacting appropriate remote service providers and downloading the client stubs or a handle for each service; d) maintaining a list of configured services; and, e) providing access to the configured services.

In a preferred embodiment a downloadable middleware called the Virtual Environment Manager (VEM) is provided. The VEM allows applications to be developed completely independently of the architecture and environment of a client computer and the servers it connects to. For a client (i.e., a network computer) to access a service, the VEM queries a Service Directory Table available on one or more connected servers. The access to the Service Directory Table returns a handle, which is used to connect to the indicated service provider.

In a preferred embodiment, a method for providing access to network services to an application executing on a network client is proposed. The method comprises the steps of determining that the application requires access to a remote network service. In response to this determination, the remote network service required by the application on demand is located, and at least one object is downloaded into a memory of the network node that enables the application to access the remote network service on demand.

Preferably, the network client is a Network Computer, the network node is a JAVA terminal or a personal digital assistant and the network service is a System Service.

In different alternate embodiments, the system service may be a printing service or persistent storage.

In other embodiments, the system service is system monitoring or system management.

BRIEF DESCRIPTION OF THE DRAWING

The present invention will be understood by reference to the drawing, wherein:

Figure 1 depicts a loosely coupled system suitable for use with the present invention;

Figure 2 show the flow chart for the boot phase of the network computer;

Figure 3 shows the flow chart for initial steps taken by each service provider;

Figure 4 shows an instance of the system state with the VEM configured;

Figure 5 shows the flow control process for an application when it wishes to use a network service; and,

Figure 6 shows an example desktop.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

A loosely coupled system suitable for use with the present invention is illustrated in FIG. 1. The system includes several computers 102-106 interconnected by way of a communication network 112. Of these computers, some are known as Network Computers 106(1), 106(2), because they rely on services available via the network to provide many of their basic functions. Others are known as service providers 104(1), 104(2), 104(3) because they provide network services, such as basic system services, to the Network Computers. Some of the computers are also known as Service Directory Managers (SDMs) 102, because they maintain a list of services that are provided to network computers by the service providers.

The Network Computers 106 can be embodied, for example, on a JAVA terminal, a personal digital assistant (PDA) or an internet terminal.

The communication protocol is HTTP and TCP/IP. The network 112 can be, for example, a token ring. The service systems 102, 104(1), 104(2), 104(3) can be embodied, for example, on IBM RISC System/6000 machines using AIX 4.2.

A logical organization can be placed on the physical system described above. This organization can be described by a three tier client/server strategy. In this strategy, tier 1 represents the client function, tier 2 represents the service provider and tier 3 represents the data object server (i.e., information storage). The system described in the present embodiment relates to tiers 1 and 2 and the interface in between. The interface in between tiers 2 and 3 is unrestricted, so users can rely on conventional, tried and true data information access systems. Note that a service provider can be designed to either reside completely on a tier 2 server or to allocate its function between tiers 1 and 2 by providing a special service stub.

Each Network Computer includes a Virtual Environment Manager (VEM) 108(1), 108(2) that is downloaded from a Service Provider by way of the communication network 112. The VEM is embodied as program code instantiated in the random access memory (not shown) of the Network Computer. In particular, the VEM provides a base class called VEMCLASS (defined in Appendix A) that it relies on for defining a collection of objects (in the sense of "object oriented programming"). One of the objects included in each VEM is an active entity called the Virtual Environment Supervisor (VES) 109(1), 109(2) that is responsible for maintaining the VEM state. In a preferred embodiment, the VES is an active JAVA object and is directly instantiated from the VEMCLASS. The VEM state includes a table of configured services 110(1), 110(2) and a table of active applications 112(2), 112(2). Both the table of configured services and the table of active applications are also instantiated in the random access memory of each Network Computer. Applications inherit from the VEMCLASS in order to interact with the VES. The object class is compiled with the application and bound to the name space of the application.

According to an embodiment of the present invention, when a network computer is switched on, it goes through a boot process in which it readies itself for use. At the end of the boot process, the network computer prompts the user for identification. The identification process can be, for example, typing in the user name and a password. After user

identity verification, the network computer presents the user with a desktop environment.

The desktop environment is preferably a JAVA container. The JAVA container runs on top of a JAVA virtual machine. Both the container and the JAVA virtual machine are provided by a program such as a Web Browser 114(1), 114(2) (e.g. Netscape Navigator) which executes on the Network Computer. The browser can be downloaded during the Network Computer's boot sequence or provided as an integral part of the Network Computer.

As part of the initialization process, the browser downloads the VES and a configuration file for the user from a Service Provider. The VES uses the configuration file to set up the user's desktop environment (desktop). An example desktop (which can be in the form of a home page) is shown in Figure 6. The configuration file contains a list of system and application services that the user accesses most frequently. These system and application services can be embodied as icons 602 or buttons 604 and/or a textual list 606 displayed on the desktop. The system and application services can be accessed as hyperlinks or as direct control buttons where the appropriate JAVA code has been downloaded. The remainder of the VEM is downloaded with user applications.

The initial logging and download of the home page is described in FIG. 2. In step 202 the Network Computer is powered on or reset by a user. In response, in step 204 the Network Computer commences its machine specific, conventional boot sequence. This sequence can be extended to include downloading of the Web browser. In step 206, the Network Computer prompts the user for a username and password. The Network Computer uses conventional methods to determine if the username and password are valid in step 208. Those of skill in the art will recognize that steps 206 and 208 could alternatively be performed as part of step 204. If step 208 determines that either the username or password are not valid, the method returns to step 206. If the username and password are valid, in step 210 the Network Computer downloads the VES and the configuration file.

When a VES is downloaded for a user, it initiates configuration of all services that are in the configuration file. The configuration file can be of a conventional flat file format. For example, the configuration file can be in the form of a Bookmarks file of the type used by Netscape Navigator 3.0. This configuration is performed by contacting the

appropriate service providers and initializing the client's table of services 110 with the server information. Stubs are also downloaded for those services that reside on both tiers 1 and 2.

In addition to the currently configured services, the VES has access to all services available on the network. This enables a user to add any available service to the desktop and as a result to the user's configuration file.

Service access is provided by a Service Directory Manager (SDM) that maintains a table of services referred to as the Service Directory Table (SDT) 114 (shown in Figure 1). The Service Directory Table contains information about system and application services offered by the Service Providers on the network. It should be understood that there can be a plurality of SDMs on the network, each maintaining a SDT or accessing a single instance of the SDT.

When a Service Provider is connected to the network, it announces the set of services it offers to the SDM. The steps taken by each Service Provider in announcing the set of services it offers are shown in Figure 3. In step 302 the Service Provider generates a list of services that it offers. In step 304, the Service Provider identifies the SDM that is responsible for maintaining the list of services in the network. In step 306 the Service Provider scans the list of services and determines the next service to be registered. If there is another service to be registered, in step 308 the Service Provider sends a REGISTER message to the SDM (directory M) and then returns to step 306. If there are no more services in the list, initialization is completed in step 310. It should be understood that steps 306 and 308 can be replaced with a single step in which the entire list is read only once and then sent to the appropriate SDM as part of a single message or message sequence.

An instance of the system state in which various client stubs are located at the Network Computers is shown in Figure 4. The Figure shows three Service Providers: a Print Server 402, a Fax Server 404 and a Mail Server 406. The SDT 408 includes information about the three service providers. In particular, the information describes the type (attribute) of service and the location of the Service Provider which provides the particular service. The particular instance of the Network Computer 410 includes various applications 412 (AP1-APn), an active VES object 414 which contains a table of configured services and passive stub objects

416 (Print Client Stub, Fax Client Stub, Mail Client Stub) that provide connections to the services. The VEM includes the VES, the stubs and the VEMCLASS portion of each application.

The VEM will now be discussed in more detail. For reference, code definitions for the VEM include the VEMCLASS and the client and server interfaces. These are is shown in appendices A and B respectively. Figure 5 shows the flow control process for an application when it wishes to use a network service. As previously discussed, loading the VES into a client is analogous to looking at an HTML page in a browser. The VES is downloaded and it's init() method is called. The init() method synchronizes itself to ensure that the class variable VE_supervisor is initialized only once. Any subsequent attempts to start a VES are disabled. The VES assigns itself id=0 by setting the VEM_id instance variable. The class variable, num_AFE, is then incremented. A registry (that contains the Configured Services Table 110 and the Table of Applications 112) and a shared_services table are created. Finally, a directory services remote object is instantiated, using the DS_server parameters passed in the html file.

Once the VEM has been downloaded and initialized, applications (APs) can be downloaded. (Note that APs are assumed to be applets in this discussion. The VEM supports applications, though applets are preferred for improved manageability.) In it's init() method, an AP should call the registerAFE() method. This call sets the AP's instance variable, VEM_id, to the class variable num_AFE, and increments num_AFE. Since num_AFE is never decremented, the AP now has a unique id. The VES is then called to create an entry for the AP in the registry (the Table of Applications 112). At this time, the entry contains only the applet object reference, id and name, but it can be expanded later as needed. The registerAFE() method returns a unique key, called a "cookie", to the AP to ensure that access to the AP information in the registry is controlled. The AP is now fully integrated into the VEM environment and is now capable of VEM function.

When the AP needs a service, it first requests the VEM to register the service using one of the registerService() methods. On service registration the VES checks the directory service (the SDM) using the LookupService() method to get the handle to the service and to see if the service has a stub. If it does, the stub is downloaded. The VEM then adds a service entry to the registry (the Configured Services Table 112)

that includes the service's handle and, if appropriate, a reference to the stub. A cookie is passed back to the AP. Once the service is in the registry, both the AP and the VES can access it as needed.

The choice of which registerService() method is called depends on: (1) whether a generic service is needed (i.e. one identifiable with just a name), or whether a more custom service is needed (e.g. one that is identified with a list of attributes as well as a name); and (2) whether the service shared. If the service is not shared, then the AP registering the service is the only one that gets access to the cookie. Otherwise, if the service is present, a cookie is passed back that is the same as those given to other APs for the service. If a shared service is not yet available in the VEM, the shared_service variable is updated to reflect the availability of a new, shared service, and a new cookie is passed back to the AP.

Access to shared services can be controlled using an access control list. Alternatively, shared services can be globally available, i.e. any AP can receive the cookie for the service just by asking for the service. Shared services are useful for minimizing the number of server stubs resident on a client. They can also be useful if service behaviour changes caused by one AP are used by another interactively.

Shared services can be removed from the VEM in various ways. According to a first embodiment, a reference count is kept for each service. The reference indicates how many APs currently have access to the service. When the reference count reaches zero, the service is removed by the VES. According to an alternative embodiment, only the AP that originally brought in the service is allowed to remove it.

Once the VEM state has been updated to include the service, the AP can get the handle by issuing the getServiceHandle() method, or get the stub by using the getServiceInstance() method. The AP is then free to make a connection of any sort and use the service. The VEM can be embodied to restrict how an AP and service communicate. When the service is no longer needed, the AP calls the RevokeService() method that updates the VEM state as follows: if the service is unique to the AP then the service is removed; if the service is a shared service then it is removed in accordance with the rules discussed above.

APPENDIX A

```
/*
 * THE VEM BASE CLASS THAT THE VES INSTANTIATES AND THAT THE
 *  APPLICATIONS INHERIT IS SHOWN BELOW.
 */

public class VEMCLASS extends Applet implements VEMBaseInterface {
  /* The VE_supervisor is a single variable available to the VES and to all applications
*/
  private static VEMCLASS VE_supervisor = null;
  /* num_AFE is a single, monotonically increasing count used for generating
application IDs */
  private static int num_AFE = 0;
  /* VEM_id is a variable per application that provides storage of an application's ID
*/
  protected int VEM_id;
  /* The registry provides the storage for the configured services table and also
application state */
  private Hashtable registry;
  /*    The shared_services variable provides a quick search mechanism for
        shared services */
  private Hashtable shared_services;
  /* The String primary, secondary DS_server variables store the location of the SDM
*/
  private String primary_DS_server, secondary_DS_server;
  /* The directory variable provides direct access to the SDM */
  private ServDir directory;

public VEMCLASS(){
}

public void init() {
}

/*
 * IMPLEMENTATION FOR THE VEMBaseInterface DESCRIBED IN
APPENDIX B
 */

public Cookie registerAFE(String name)
  throws VEMRegistryException {
}

public Cookie registerService(String name)
  throws VEMRegistryException {
}

public Cookie registerService(String name,String[] attrs)
  throws VEMRegistryException {
}
```

```
public Cookie registerService(String name, boolean shared)
  throws VEMRegistryException {
}

public Cookie registerService(String name,String[] attrs, boolean shared)
  throws VEMRegistryException {
}

public ServiceHandle getServiceHandle(Cookie c)
  throws VEMRegistryException {
}

public Object getServiceInstance(Cookie c)
  throws VEMRegistryException {
}

public boolean revokeService(Cookie c)
  throws VEMRegistryException {
}

/*
 * INTERNAL ROUTINES TO DIRECTLY MANAGE THE VES STATE
 */

synchronized private void addToRegistry(Cookie c, RegistryElement r)
  throws VEMRegistryException {
}

synchronized private RegistryElement getFromRegistry(Cookie c) {
}

synchronized private String removeFromRegistry(Cookie c)
  throws VEMRegistryException{
}

synchronized private Cookie shareService(int o, String name) {
}

synchronized private void removeIfShared(int o, String name) {
}
}
```

APPENDIX B

```
/* THE VEM BASE INTERFACE */

public interface VEMBaseInterface {
  public static final boolean SHARED = true;

  /*-------------------------------------------------------
   | Methods to notify VEMSupervisor of resident application front ends. A  |
   | unique key is passed back to allow only the application and VEMSupervisor |
   | access to any VEM state information of the application.                |
   -------------------------------------------------------*/

  /*
   | Register the Application Front End (AFE) with the VEM
   */
  public Cookie registerAFE(String name)
     throws VEMRegistryException ;

  /*-------------------------------------------------------
   | Methods to notify VEMSupervisor of required services needed by an AFE.  |
   | This call updated VEM state with a handle to a service that the AFE can  |
   | dynamically access. The AFE sets up communication with the service      |
   | using the handle. A unique key is passed back, allowing the AFE         |
   | exclusive access to the service handle.                                 |
   -------------------------------------------------------*/

  /*
   | Set up the VEM state with a handle for any service with the name "name".
   */
  public Cookie registerService(String name)
     throws VEMRegistryException ;

  /*
   | Set up the VEM state with a handle for a service with the name "name",
   | and containing the attributes "attrs".
   */
  public Cookie registerService(String name,String[] attrs)
     throws VEMRegistryException ;
```

```
/*
 | If shared is equal to "SHARED", i.e., true, then see if service with the
 | name "name" is known to the VEM.  If it is not, set up the VEM state
 | with the shared service and set the owner to "this" AFE. Return the
 | associated key.
 */
public Cookie registerService(String name, boolean shared)
    throws VEMRegistryException ;

/*
 | If shared is equal to "SHARED", i.e., true, then see if service with the
 | name "name", and attributes "attrs" is known to the VEM.  If it is not,
 | set up the VEM state with the shared service and set the owner to "this"
 | AFE. Return the associated key.
 */
public Cookie registerService(String name,String[] attrs, boolean shared)
    throws VEMRegistryException ;

/*-------------------------------------------------------------
 | Methods for AFE to get access to services known to VEM      |
 -------------------------------------------------------------*/

/*
 | Get the handle associated with a service that can be used to set up
 | a connection with the service
 */
public ServiceHandle getService(Cookie c)
    throws VEMRegistryException;

/*
 | Get access to a service stub, often used for shared services.
 */
public Object getServiceStub(Cookie c)
    throws VEMRegistryException;

/*-------------------------------------------------------------
 | Method to remove a service from all VEM state.              |
 -------------------------------------------------------------*/
```

```
    /*
     | Remove the service identified with the key.  If the service is shared,
     | only the owner AFE is allowed to remove it.
     */
    public boolean revokeService(Cookie c)
        throws VEMRegistryException ;
}

/* THE SDM INTERFACE */

public interface VEM_ServDir extends java.rmi.Remote {

    /*-------------------------------------------------------------
    | Methods to lookup services. A handle is returned for a service even if    |
    | it is marked "failed". The client decdies what to do with it. The returned |
    | handle looks like {<ip_address>:<port #>:<flag>}. The flag may be "failed" |
    | or "alive".                                                               |
    -------------------------------------------------------------*/

    /*
     | Return service handle for the first service that matches service name
     */
    String lookUpService (String ServiceName)
        throws java.rmi.RemoteException;

    /*
     | Return service handle for the first service that matches service name and
     | all attributes
     */
    String lookUpService (String ServiceName, String[] ServiceAttr)
        throws java.rmi.RemoteException;

    /*
     | Return service handle for the "next" service that matches service name and
     | that comes "after" the supplied service handle
     */
    String lookUpService (String ServiceName, String ServiceHandle)
        throws java.rmi.RemoteException;
```

```
/*
 | Return service handle for the "next"service that matches service name and
 | all attributes and that comes "after" the supplied service handle
 */
String lookUpService (String ServiceName, String ServiceHandle,
                      String[] ServiceAttr)
   throws java.rmi.RemoteException;


/*-----------------------------------------------------------------
| Methods to get service counts. Counted also if marked "failed".  |
-----------------------------------------------------------------*/

/*
 | Return count for matching service names
 */
int getServiceCount (String ServiceName)
                            throws java.rmi.RemoteException;

/*
 | Return count for matching service names and attributes
 */
int getServiceCount (String ServiceName, String[] ServiceAttr)
   throws java.rmi.RemoteException;


/*-----------------------------------------------------------------
| Methods that return meta information about services. Meta information  |
| includes "failed" as well as "alive" services                          |
-----------------------------------------------------------------*/

/*
 | Return an array of strings. The first element is the name of the service,
 | the rest are attributes. Input is service handle.
 */
string[] getServiceNameAttr (string ServiceHandle)
   throws java.rmi.RemoteException;
```

```
        /*
         | Return an array of strings. Each element of that array contains attributes
         | of the service separated by special delimiter "$". We stipulate that
         | service names, attributes, and service handles not contain "$".
         */
        string[] getServiceAttr (string ServiceName)
            throws java.rmi.RemoteException;

        /*
         | Return an array of strings. Each element of that array contains first
         | the service name, followed by service attributes. All are delimited by
         | the "$" character.
         */
        string[] getListOfServices ()
            throws java.rmi.RemoteException;
    }
```

CLAIMS

1. A method for providing access to network services to an application executing on a network client, comprising the steps of:

determining when the application requires access to a remote network service;

in response to a determination that the application requires access to the remote network service: locating the remote network service required by the application on demand; and downloading at least one object into a memory of the network node that enables the application to access the remote network service on demand.

2. The method of Claim 1 wherein the downloading comprises the step of contacting an appropriate service provider and downloading a client stubs for the remote network service.

3. The method of Claim 1 wherein the downloading comprises the step of contacting an appropriate service provider and downloading a client handle for the remote network service.

4. The method of any preceding Claim comprising the further step of providing a user of the network client with a visual display of available network services to which access has been previously enabled.

5. The method of any preceding Claim wherein a virtual environment manager running on a virtual machine instantiated on the network node, provides access to the network services.

6. The method of any preceding Claim comprising the further step of determining when access to the network service is no longer needed; and, initiating a removal of the object from the memory of the network node.

7. The method of any preceding Claim comprising the further steps of registering by remote service providers with a third party directory, a set of services available to network clients, wherein the object obtains access to the service by reference to the directory.

8. The method of any preceding Claim comprising the further step of sharing access to the service by a plurality of application executing on the network node.

9. The method of any preceding Claim comprising the further step of isolating the service needed by an application from other applications by providing access to the service only by way of a unique key known only to the application.

10. A method for providing on-demand access to system services to applications executing on a network computer, comprising the steps of:

a) in response to a demand from a user, downloading a virtual environment manager that provides access to remote network services;

b) identifying a set of system services;

c) configuring the set of system services by contacting appropriate remote service providers and downloading the client stubs or a handle for each service;

d) maintaining a list of configured services; and,

e) providing access to the configured services.

11. The method of Claim 10 comprising the further step of sharing access to the system services by a plurality of the applications.

The Patent Office

18

Application No: GB 9720400.2 **Examiner:** Melanie Gee
Claims searched: 1-11 **Date of search:** 7 April 1998

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.P): G4A (AFGN, AUDB)

Int Cl (Ed.6): G06F 13/14, 17/30

Other: Online: WPI, INSPEC, COMPUTER

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
X	EP 0780778 A2	(SUN MICROSYSTEMS), see col. 7 line 45 - col. 9 line 26	1
X	US 5483652 A	(SUDAMA et al.), see whole document	1

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.